**品优购电商系统开发**

**第13章**

**消息中间件解决方案JMS**

传智播客.黑马程序员

# 课程目标

目标1：理解消息中间件、JMS等概念

目标2：掌握JMS点对点与发布订阅模式的收发消息

目标3：掌握SpringJMS集成

目标4：完成商品审核导入索引库

目标5：完成商品删除移除索引库

目标6: 完成商品审核生成商品详细页

目标7: 完成商品删除完成删除商品详细页

# 1. JMS入门

## 1.1消息中间件

### 1.1.1现阶段的服务调用存在的问题

服务和服务之间的耦合度很高。在商品审核时，同时

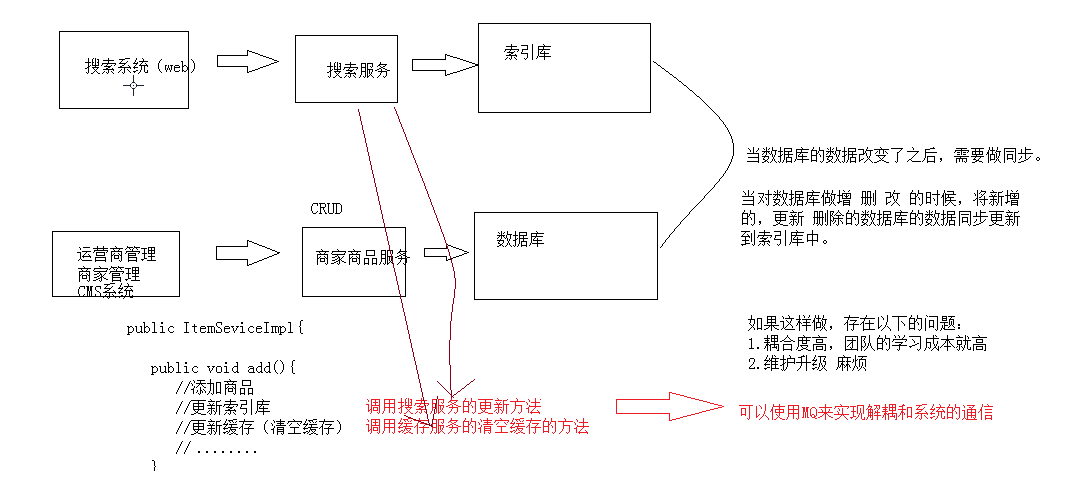
需要调用商品服务，

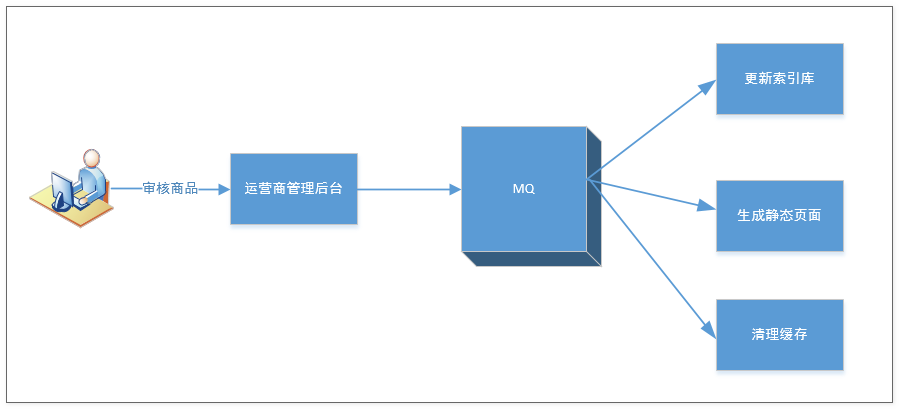
需要调用生成静态页面服务，

需要调用更新索引库的服务。

随着业务的增加耦合度提升。服务之间的启动先后顺序变的复杂。所以不利于扩展和维护。

使用消息的中间件来解决这个问题。





### 1.1.2什么是消息中间件（MQ）

message queue

消息中间件利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行[分布式系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E7%B3%BB%E7%BB%9F)的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。对于消息中间件，常见的角色大致也就有Producer（生产者）、Consumer（消费者）

常见的消息中间件产品:

**（1）ActiveMQ**

ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现。我们在本次课程中介绍 ActiveMQ的使用。

ActiveMQ 是apache出品的一个实现了JMS规范的MQ的其中一个产品.主要是用于消息发送和接收来实现系统之间的通信

（2）RabbitMQ(ERLANG)

AMQP协议的领导实现，支持多种场景。淘宝的MySQL集群内部有使用它进行通讯，OpenStack开源云平台的通信组件，最先在金融行业得到运用。

（3）ZeroMQ

史上最快的消息队列系统

（4）Kafka

Apache下的一个子项目 。特点：高吞吐，在一台普通的服务器上既可以达到10W/s的吞吐速率；完全的分布式系统。适合处理海量数据。

MQ:应用场景：解耦系统，异构系统 ，抢购 流量限制。流量削峰

（MQ:用于更新索引库 生成静态页面，订单的短信推送，清除缓存）。

### 1.1.3改造系统模块调用关系

我们通过引入消息中间件activeMQ,使得运营商系统与搜索服务、页面生成服务解除了耦合。

## 1.2 JMS简介

### 1.2.1什么是JMS

JMS（[**Java**](http://lib.csdn.net/base/java)Messaging Service）是Java平台上有关面向消息中间件的技术规范，它便于消息系统中的Java应用程序进行消息交换,并且通过提供标准的产生、发送、接收消息的接口简化企业应用的开发。

       JMS本身只定义了一系列的接口规范，是一种与厂商无关的 API，用来访问消息收发系统。它类似于 JDBC([**java**](http://lib.csdn.net/base/java)Database Connectivity)：这里，JDBC 是可以用来访问许多不同关系[**数据库**](http://lib.csdn.net/base/mysql)的 API，而 JMS 则提供同样与厂商无关的访问方法，以访问消息收发服务。许多厂商目前都支持 JMS，包括 IBM 的 MQSeries、BEA的 Weblogic JMS service和 Progress 的 SonicMQ，这只是几个例子。 JMS 使您能够通过消息收发服务（有时称为消息中介程序或路由器）从一个 JMS 客户机向另一个 JML 客户机发送消息。消息是 JMS 中的一种类型对象，由两部分组成：报头和消息主体。报头由路由信息以及有关该消息的元数据组成。消息主体则携带着应用程序的数据或有效负载。

JMS 定义了五种不同的消息正文格式，以及调用的消息类型，允许你发送并接收以一

些不同形式的数据，提供现有消息格式的一些级别的兼容性。

· TextMessage--一个字符串对象

· MapMessage--一套名称-值对

· ObjectMessage--一个序列化的 Java 对象

· BytesMessage--一个字节的数据流

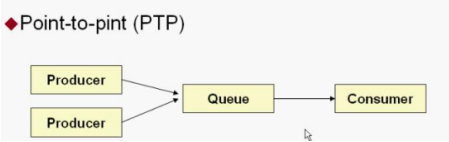
· StreamMessage -- Java 原始值的数据流

### 1.2.2 JMS消息传递类型

对于消息的传递有两种类型：

一种是点对点的，即一个生产者和一个消费者一一对应；

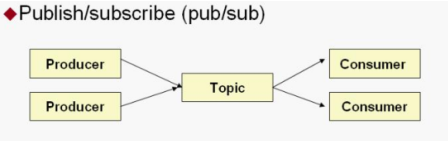
点对点：一个生产者产生一个消息，只能被一个消费者消费。



另一种是发布/ 订阅模式，即一个生产者产生消息并进行发送后，可以由多个消费者进

行接收。

前提：先要订阅topic 才能接受到消息。



## 1.3ActiveMQ下载与安装

### 1.3.1下载

官方网站下载：<http://activemq.apache.org/>

### 1.3.2安装（Linux）

1. 将apache-activemq-5.12.0-bin.tar.gz 上传至服务器
2. 解压此文件

|  |
| --- |
| tar zxvf apache-activemq-5.12.0-bin.tar.gz |

1. 为apache-activemq-5.12.0目录赋权

|  |
| --- |
| chmod 777 apache-activemq-5.12.0 |

1. 进入apache-activemq-5.12.0\bin目录

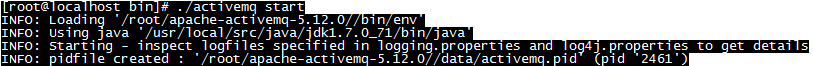
（5）赋与执行权限

|  |
| --- |
| chmod 755 activemq  -------------------------------------- **知识点小贴士** --------------------------  linux 命令chmod 755的意思  chmod是[**Linux**](http://lib.csdn.net/base/linux)下设置文件权限的命令，后面的数字表示不同用户或用户组的权限。  一般是三个数字： 第一个数字表示文件所有者的权限 第二个数字表示与文件所有者同属一个用户组的其他用户的权限 第三个数字表示其它用户组的权限。        权限分为三种：读（r=4），写（w=2），执行（x=1） 。 综合起来还有可读可执行（rx=5=4+1）、可读可写（rw=6=4+2）、可读可写可执行(rwx=7=4+2+1)。        所以，chmod 755 设置用户的权限为：  1.文件所有者可读可写可执行                                   --7  2.与文件所有者同属一个用户组的其他用户可读可执行 --5  3.其它用户组可读可执行                                        --5 |

### 1.3.3启动

|  |
| --- |
| ./activemq start |

出现下列提示表示成功！



假设服务器地址为192.168.25.135 ，打开浏览器输入地址

[http://192.168.25.135:8161/](http://192.168.25.129:8161/) 即可进入ActiveMQ管理页面



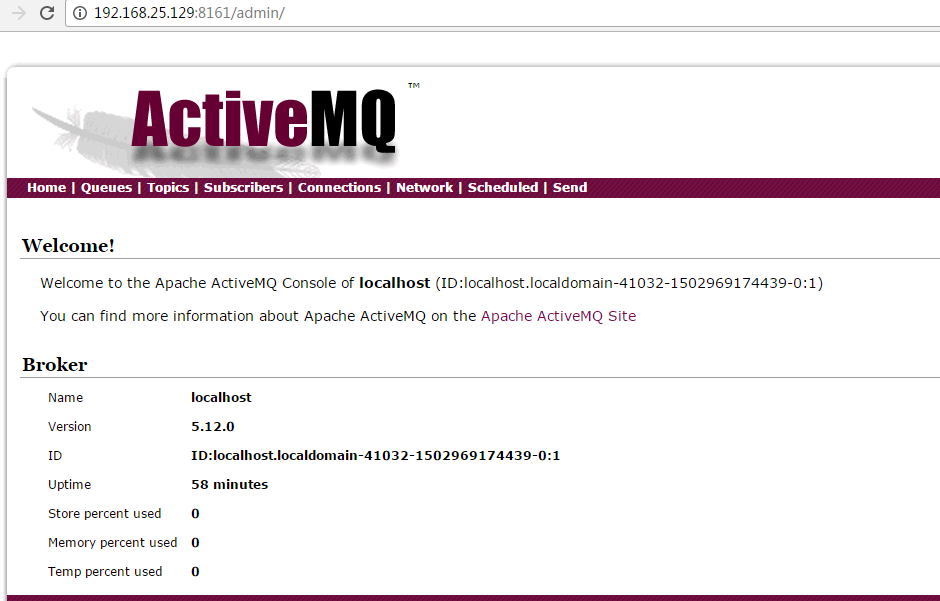
点击进入管理页面



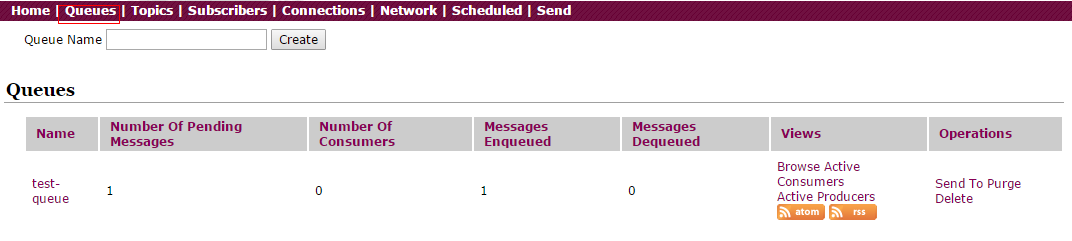
输入用户名和密码 均为 admin



进入主界面



点对点消息列表：



列表各列信息含义如下：

[**Number Of Pending Messages**](http://192.168.25.135:8161/admin/queues.jsp)**：**等待消费的消息 这个是当前未出队列的数量。

[**Number Of Consumers**](http://192.168.25.135:8161/admin/queues.jsp)**：**消费者 这个是消费者端的消费者数量

[**Messages Enqueued**](http://192.168.25.135:8161/admin/queues.jsp)**：**进入队列的消息  进入队列的总数量,包括出队列的。

[**Messages Dequeued**](http://192.168.25.135:8161/admin/queues.jsp)**：**出了队列的消息  可以理解为是消费这消费掉的数量。

# 2. JMS入门

## 2.1点对点模式

一个生成者产生一个消息 只能被被一个消费者消费，消费完，消息就没有了。

### 2.1.1消息生产者

（1）创建工程jmsDemo ，引入依赖

|  |
| --- |
| <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-client</artifactId>  <version>5.13.4</version>  </dependency> |

（2）创建类QueueProducer main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.25.133:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建队列对象  Queue queue = session.createQueue("test-queue");  //6.创建消息生产者  MessageProducer producer = session.createProducer(queue);  //7.创建消息  TextMessage textMessage = session.createTextMessage("欢迎来到神奇的品优购世界");  //8.发送消息  producer.send(textMessage);  //9.关闭资源  producer.close();  session.close();  connection.close(); |

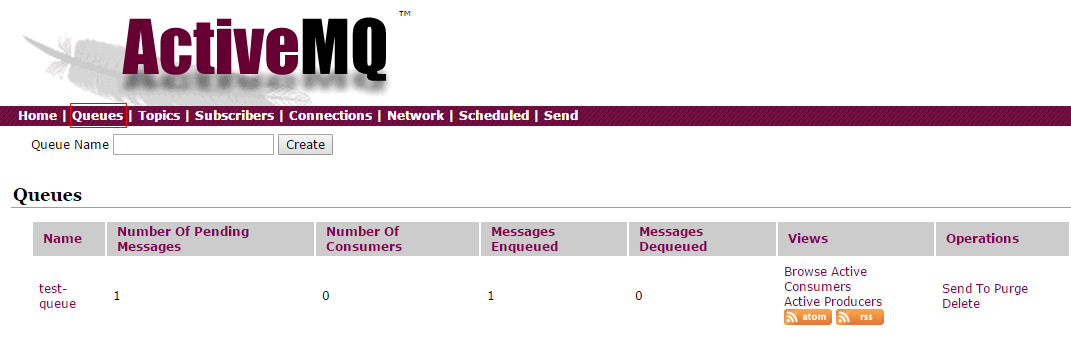
上述代码中第4步创建session 的两个参数：

第1个参数 是否使用事务

第2个参数 消息的确认模式

* AUTO\_ACKNOWLEDGE = 1    自动确认
* CLIENT\_ACKNOWLEDGE = 2    客户端手动确认
* DUPS\_OK\_ACKNOWLEDGE = 3    自动批量确认
* SESSION\_TRANSACTED = 0    事务提交并确认

运行后通过ActiveMQ管理界面查询



### 2.1.2消息消费者

创建类QueueConsumer ，main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.25.133:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建队列对象  Queue queue = session.createQueue("test-queue");  //6.创建消息消费  MessageConsumer consumer = session.createConsumer(queue);    //7.监听消息  consumer.setMessageListener(**new** MessageListener() {  **publicvoid** onMessage(Message message) {  TextMessage textMessage=(TextMessage)message;  **try** {  System.***out***.println("接收到消息："+textMessage.getText());  } **catch** (JMSException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  });  //8.等待键盘输入  System.***in***.read();  //9.关闭资源  consumer.close();  session.close();  connection.close(); |

执行后可以看到控制台输出

### 2.1.3运行测试

同时开启2个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现只有一个消费者会接收到消息。

## 2.2 发布/订阅模式

### 2.2.1消息生产者

创建类TopicProducer ，main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.25.133:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建主题对象  Topic topic = session.createTopic("test-topic");  //6.创建消息生产者  MessageProducer producer = session.createProducer(topic);  //7.创建消息  TextMessage textMessage = session.createTextMessage("欢迎来到神奇的品优购世界");  //8.发送消息  producer.send(textMessage);  //9.关闭资源  producer.close();  session.close();  connection.close(); |

运行效果如下：



### 2.2.2消息消费者

创建类TopicConsumer ,main方法代码如下：

|  |
| --- |
| //1.创建连接工厂  ConnectionFactory connectionFactory=**new** ActiveMQConnectionFactory("tcp://192.168.25.135:61616");  //2.获取连接  Connection connection = connectionFactory.createConnection();  //3.启动连接  connection.start();  //4.获取session (参数1：是否启动事务,参数2：消息确认模式)  Session session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);  //5.创建主题对象  //Queue queue = session.createQueue("test-queue");  Topic topic = session.createTopic("test-topic");  //6.创建消息消费  MessageConsumer consumer = session.createConsumer(topic);    //7.监听消息  consumer.setMessageListener(**new** MessageListener() {  **publicvoid** onMessage(Message message) {  TextMessage textMessage=(TextMessage)message;  **try** {  System.***out***.println("接收到消息："+textMessage.getText());  } **catch** (JMSException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  });  //8.等待键盘输入  System.***in***.read();  //9.关闭资源  consumer.close();  session.close();  connection.close(); |

### 2.2.3运行测试

同时开启2个以上的消费者，再次运行生产者，观察每个消费者控制台的输出，会发现每个消费者会接收到消息。

总结：

*//发布订阅的模式 默认的请情况下：消息的内容不存在服务器，当生产者发送了一个消息，如果消费者之前没有订阅，就没了。一个生产者产生一个消息会被多个消费者消费。  
//点对点的方式：默认的请情况下：将消息存储在服务器上，消费者随时来取，但是一旦一个消费者获取到了消息，这个消息就没有了。*

# 3.Spring整合JMS

## 3.1 点对点模式

### 3.1.1消息生产者

1. 创建工程springjms\_producer，在POM文件中引入SpringJms 、activeMQ以及单元测试相关依赖

（2）在src/main/resources下创建spring配置文件applicationContext-jms-producer.xml

|  |
| --- |
| <context:component-scanbase-package=*"cn.itcast.demo"*></context:component-scan>  <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <beanid=*"targetConnectionFactory"*class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <propertyname=*"brokerURL"*value=*"tcp://192.168.25.135:61616"*/>  </bean>  <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <beanid=*"connectionFactory"*class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <propertyname=*"targetConnectionFactory"*ref=*"targetConnectionFactory"*/>  </bean>  <!-- Spring提供的JMS工具类，它可以进行消息发送、接收等 -->  <beanid=*"jmsTemplate"*class=*"org.springframework.jms.core.JmsTemplate"*>  <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  </bean>  <!--这个是队列目的地，点对点的 文本信息-->  <beanid=*"queueTextDestination"*class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-argvalue=*"queue\_text"*/>  </bean> |

（3）在cn.itcast.demo包下创建消息生产者类

|  |
| --- |
| @Component  **publicclass** QueueProducer {    @Autowired  **private** JmsTemplate jmsTemplate;    @Autowired  **private** Destination queueTextDestination;    /\*\*  \* 发送文本消息  \* **@param** text  \*/  **publicvoid** sendTextMessage(**final** String text){  jmsTemplate.send(queueTextDestination, **new** MessageCreator() {  **public** Message createMessage(Session session) **throws** JMSException {  **return**session.createTextMessage(text);  }  });  }  } |

（4）单元测试

在src/test/java创建测试类

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations="classpath:applicationContext-jms-producer.xml")  **publicclass** TestQueue {  @Autowired  **private** QueueProducer queueProducer;    @Test  **publicvoid** testSend(){  queueProducer.sendTextMessage("SpringJms-点对点");  }  } |

### 3.1.2消息消费者

1. 创建工程springjms\_consumer，在POM文件中引入依赖 （同上一个工程）

（2）创建配置文件 applicationContext-jms-consumer-queue.xml

|  |
| --- |
| <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <beanid=*"targetConnectionFactory"*class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <propertyname=*"brokerURL"*value=*"tcp://192.168.25.135:61616"*/>  </bean>  <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <beanid=*"connectionFactory"*class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <propertyname=*"targetConnectionFactory"*ref=*"targetConnectionFactory"*/>  </bean>  <!--这个是队列目的地，点对点的 文本信息-->  <beanid=*"queueTextDestination"*class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-argvalue=*"queue\_text"*/>  </bean>  <!-- 我的监听类 -->  <beanid=*"myMessageListener"*class=*"cn.itcast.demo.MyMessageListener"*></bean>  <!-- 消息监听容器 -->  <beanclass=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  <propertyname=*"destination"*ref=*"queueTextDestination"*/>  <propertyname=*"messageListener"*ref=*"myMessageListener"*/>  </bean> |

（3）编写监听类

|  |
| --- |
| **publicclass** MyMessageListener **implements** MessageListener {  **publicvoid** onMessage(Message message) {  TextMessage textMessage=(TextMessage)message;  **try** {  System.***out***.println("接收到消息："+textMessage.getText());  } **catch** (JMSException e) {  e.printStackTrace();  }  }  } |

（4）创建测试类

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations="classpath:applicationContext-jms-consumer-queue.xml")  **publicclass** TestQueue {  @Test  **publicvoid** testQueue(){  **try** {  System.***in***.read();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

## 3.2发布/订阅模式

### 3.2.1消息生产者

（1）在工程springjms\_producer的applicationContext-jms-producer.xml增加配置

|  |
| --- |
| <!--这个是订阅模式 文本信息-->  <beanid=*"topicTextDestination"*class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-argvalue=*"topic\_text"*/>  </bean> |

（2）创建生产者类

|  |
| --- |
| @Component  **publicclass** TopicProducer {  @Autowired  **private** JmsTemplate jmsTemplate;    @Autowired  **private** Destination topicTextDestination;    /\*\*  \* 发送文本消息  \* **@param** text  \*/  **publicvoid** sendTextMessage(**final** String text){  jmsTemplate.send(topicTextDestination, **new** MessageCreator() {  **public** Message createMessage(Session session) **throws** JMSException {  **return**session.createTextMessage(text);  }  });  }  } |

（3）编写测试类

|  |
| --- |
| **import** org.junit.Test;  **import** org.junit.runner.RunWith;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.test.context.ContextConfiguration;  **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  **import** cn.itcast.demo.TopicProducer;  @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations="classpath:applicationContext-activemq-producer.xml")  **publicclass** TestTopic {  @Autowired  **private** TopicProducer topicProducer;  @Test  **publicvoid** sendTextQueue(){  topicProducer.sendTextMessage();  }  } |

### 3.2.2消息消费者

1. 在activemq-spring-consumer工程中创建配置文件applicationContext-jms-consumer-topic.xml

|  |
| --- |
| <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <beanid=*"targetConnectionFactory"*class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <property name=*"brokerURL"* value=*"tcp://192.168.25.133:61616"*/>  </bean>  <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <bean id=*"connectionFactory"*class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <propertyname=*"targetConnectionFactory"*ref=*"targetConnectionFactory"*/>  </bean>  <!--这个是队列目的地，点对点的 文本信息-->  <bean id=*"topicTextDestination"* class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-arg value=*"topic\_text"*/>  </bean>  <!-- 我的监听类 -->  <beanid=*"myMessageListener"*class=*"cn.itcast.demo.MyMessageListener"*></bean>  <!-- 消息监听容器 -->  <beanclass=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  <propertyname=*"destination"*ref=*"topicTextDestination"*/>  <propertyname=*"messageListener"*ref=*"myMessageListener"*/>  </bean> |

1. 编写测试类

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations="classpath:applicationContext-jms-consumer-topic.xml")  **publicclass** TestTopic {  @Test  **publicvoid**testTopic(){  **try** {  System.***in***.read();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

测试：同时运行三个消费者工程，在运行生产者工程，查看三个消费者工程的控制台输出。

# 4.商品审核-更新Solr索引库

## 4.1需求分析

运用消息中间件activeMQ实现运营商后台与搜索服务的零耦合。运营商执行商品审核后，向activeMQ发送消息（SKU列表），搜索服务从activeMQ接收到消息并导入到solr索引库。

## 4.2消息生产者（运营商后台）

### 4.2.1解除耦合

修改pinyougou-manager-web，移除搜索服务接口依赖：

|  |
| --- |
| <**dependency**>  <**groupId**>com.pinyougou</**groupId**>  <**artifactId**>pinyougou-search-interface</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**> </**dependency**> |

GoodsController.java中删除调用搜索服务接口的相关代码

|  |
| --- |
| //itemSearchService.deleteByGoodsIds(Arrays.asList(ids));  //@Reference  //private ItemSearchService itemSearchService;  //itemSearchService.importList(itemList); |

### 4.2.2准备工作

（1）修改pinyougou-manager-web的pom.xml,引入依赖

|  |
| --- |
| <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-client</artifactId>  <version>5.13.4</version>  </dependency> |

1. pinyougou-manager-web工程添加applicationContext-jms-producer.xml

改名为spring-activemq.xml 内容如下：

|  |
| --- |
| <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <beanid=*"targetConnectionFactory"*class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <propertyname=*"brokerURL"*value=*"tcp://192.168.25.135:61616"*/>  </bean>  <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <beanid=*"connectionFactory"*class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <!-- 目标ConnectionFactory对应真实的可以产生JMS Connection的ConnectionFactory -->  <propertyname=*"targetConnectionFactory"*ref=*"targetConnectionFactory"*/>  </bean>  <!-- Spring提供的JMS工具类，它可以进行消息发送、接收等 -->  <beanid=*"jmsTemplate"*class=*"org.springframework.jms.core.JmsTemplate"*>  <!-- 这个connectionFactory对应的是我们定义的Spring提供的那个ConnectionFactory对象 -->  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  </bean>  <!--这个是队列目的地，点对点-->  <beanid=*"queueSolrDestination"*class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-argvalue=*"pinyougou\_queue\_solr"*/>  </bean> |

（3）修改web.xml

|  |
| --- |
| <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/spring-\*.xml</param-value>  </context-param> |

### 4.2.3代码实现

修改pinyougou-manager-web工程的GoodsController.java

|  |
| --- |
| @Autowired  **private** Destination queueSolrDestination;//用于发送solr导入的消息  @Autowired  **private** JmsTemplate jmsTemplate;  @RequestMapping("/updateStatus")  **public** Result updateStatus(Long[] ids,String status){  **try** {  goodsService.updateStatus(ids, status);  //按照SPU ID查询 SKU列表(状态为1)  **if**(status.equals("1")){//审核通过  List<TbItem> itemList = goodsService.findItemListByGoodsIdandStatus(ids, status);  //调用搜索接口实现数据批量导入  **if**(itemList.size()>0){  **final** String jsonString = JSON.*toJSONString*(itemList);  jmsTemplate.send(queueSolrDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  **return** session.createTextMessage(jsonString);  }  });  }**else**{  System.***out***.println("没有明细数据");  }  }  **returnnew** Result(**true**, "修改状态成功");  } **catch** (Exception e) {  e.printStackTrace();  **returnnew** Result(**false**, "修改状态失败");  }  } |

## 4.3消息消费者（搜索服务）

### 4.3.1准备工作

（1）修改pinyougou-search-service ，在pom.xml中添加activemq依赖

|  |
| --- |
| <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-client</artifactId>  <version>5.13.4</version>  </dependency> |

（2）添加spring配置文件applicationContext-jms-consumer.xml

|  |
| --- |
| <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <beanid=*"targetConnectionFactory"*class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <propertyname=*"brokerURL"*value=*"tcp://192.168.25.135:61616"*/>  </bean>  <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <beanid=*"connectionFactory"*class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <propertyname=*"targetConnectionFactory"*ref=*"targetConnectionFactory"*/>  </bean>  <!--这个是队列目的地，点对点的 文本信息-->  <beanid=*"queueSolrDestination"*class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-argvalue=*"pinyougou\_queue\_solr"*/>  </bean>  <!-- 消息监听容器 -->  <beanclass=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  <propertyname=*"destination"*ref=*"queueSolrDestination"*/>  <propertyname=*"messageListener"*ref=*"itemSearchListener"*/>  </bean> |

### 4.3.2代码实现

在pinyougou-search-service的com.pinyougou.search.service.impl新增监听类

|  |
| --- |
| @Component  **public class** ItemSearchListener **implements** MessageListener{  @Autowired  **private** ItemSearchService itemSearchService;    @Override  **publicvoid** onMessage(Message message) {  System.***out***.println("监听接收到消息...");  **try** {  TextMessage textMessage=(TextMessage)message;  String text = textMessage.getText();  List<TbItem>list = JSON.*parseArray*(text,TbItem.**class**);  **for**(TbItem item:list){  System.***out***.println(item.getId()+""+item.getTitle());  Map specMap= JSON.*parseObject*(item.getSpec());//将spec字段中的json字符串转换为map  item.setSpecMap(specMap);//给带注解的字段赋值  }  itemSearchService.importList(list);//导入  System.***out***.println("成功导入到索引库");  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

# 5.商品删除-移除Solr索引库记录

## 5.1需求分析

通过消息中间件实现在商品删除时也同时移除索引库记录的功能。

## 5.2消息生产者（运营商后台）

### 5.2.1配置文件

修改pinyougou-manager-web工程的spring-activemq.xml，添加bean配置

|  |
| --- |
| <!--这个是队列目的地，点对点-->  <bean id=*"queueSolrDeleteDestination"*class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-argvalue=*"pinyougou\_queue\_solr\_delete"*/>  </bean> |

### 5.2.2代码实现

修改GoodsController.java

|  |
| --- |
| @Autowired  **private** Destination queueSolrDeleteDestination;//用户在索引库中删除记录  /\*\*  \* 批量删除  \* **@param** ids  \* **@return**  \*/  @RequestMapping("/delete")  **public** Result delete(**final** Long [] ids){  **try** {  goodsService.delete(ids);  jmsTemplate.send(queueSolrDeleteDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  **return** session.createObjectMessage(ids);  }  });  **return new** Result(**true**, "删除成功");  } **catch** (Exception e) {  e.printStackTrace();  **returnnew** Result(**false**, "删除失败");  }  } |

## 5.3消息消费者（搜索服务）

### 5.3.1配置文件

修改pinyougou-search-service的applicationContext-activemq-consumer.xml

|  |
| --- |
| <!--这个是队列目的地，点对点的 文本信息 (删除索引库中记录) -->  <beanid=*"queueSolrDeleteDestination"*class=*"org.apache.activemq.command.ActiveMQQueue"*>  <constructor-argvalue=*"pinyougou\_queue\_solr\_delete"*/>  </bean>  <!-- 消息监听容器(删除索引库中记录) -->  <beanclass=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  <propertyname=*"destination"*ref=*"queueSolrDeleteDestination"*/>  <propertyname=*"messageListener"*ref=*"itemDeleteListener"*/>  </bean> |

### 5.3.2代码实现

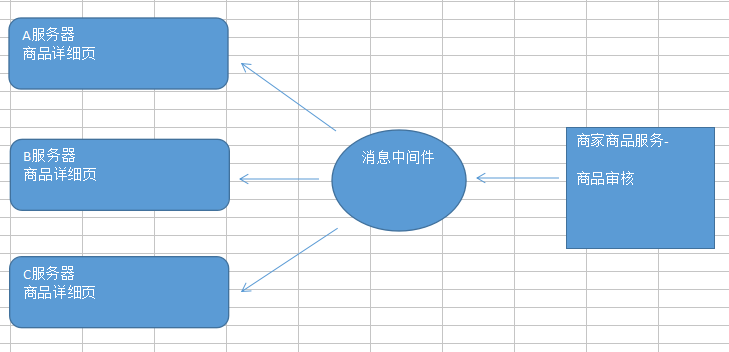
com.pinyougou.search.service.impl包下创建监听类ItemDeleteListener

|  |
| --- |
| /\*\*  \* 监听：用于删除索引库中记录  \* **@author** Administrator  \*  \*/  @Component  **publicclass** ItemDeleteListener **implements** MessageListener{  @Autowired  **private** ItemSearchService itemSearchService;    @Override  **public void** onMessage(Message message) {  **try** {  ObjectMessage objectMessage= (ObjectMessage)message;  Long[] goodsIds = (Long[]) objectMessage.getObject();  System.***out***.println("ItemDeleteListener监听接收到消息..."+goodsIds);  itemSearchService.deleteByGoodsIds(Arrays.*asList*(goodsIds));  System.***out***.println("成功删除索引库中的记录");  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

# 6.商品审核-执行网页静态化

## 6.1需求分析

运用消息中间件activeMQ实现运营商后台与网页生成服务的零耦合。运营商执行商品审核后，向activeMQ发送消息（商品ID），网页生成服务从activeMQ接收到消息后执行网页生成操作。



## 6.2消息生产者（运营商后台）

### 6.2.1解除耦合

修改pinyougou-manager-web，移除网页生成服务接口依赖：

|  |
| --- |
| <**dependency**>  <**groupId**>com.pinyougou</**groupId**>  <**artifactId**>pinyougou-page-interface</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**> </**dependency**> |

GoodsController.java中删除调用网页生成服务接口的相关代码

|  |
| --- |
| //private ItemPageService itemPageService;  //静态页生成  //for(Long goodsId:ids){  //itemPageService.genItemHtml(goodsId);  //} |

### 6.2.2准备工作

修改配置文件spring-activemq.xml，添加配置

|  |
| --- |
| <!--这个是订阅模式 文本信息-->  <beanid=*"topicPageDestination"*class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-argvalue=*"pinyougou\_topic\_page"*/>  </bean> |

### 6.2.3代码实现

修改pinyougou-manager-web的GoodsController.java

|  |
| --- |
| **public** Result updateStatus(Long[] ids,String status){  **try** {  .......  **if**(status.equals("1")){//审核通过  ........  //静态页生成  **for**(**final** Long goodsId:ids){  jmsTemplate.send(topicPageDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  **return** session.createTextMessage(goodsId+"");  }  });  }  }  ......  } **catch** (Exception e) {  ......  }  } |

## 6.3消息消费者（页面生成服务）

### 6.3.1解除dubbox依赖

1. 修改工程pinyougou-page-service ，删除dubbox相关依赖

|  |
| --- |
| <!-- dubbo相关 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>dubbo</artifactId>  </dependency>  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  </dependency>  <dependency>  <groupId>com.github.sgroschupf</groupId>  <artifactId>zkclient</artifactId>  </dependency> |

1. 修改applicationContext-service.xml，删除dubbox相关配置

|  |
| --- |
| <dubbo:protocol name=*"dubbo"* port=*"20885"*></dubbo:protocol>  <dubbo:application name=*"pinyougou-page-service"*/>  <dubbo:registry address=*"zookeeper://192.168.25.135:2181"*/>  <dubbo:annotation package=*"com.pinyougou.page.service.impl"* /> |

1. 修改ItemPageServiceImpl类的@Service注解 为org.springframework.stereotype.Service包下的@Service注解

### 6.3.2准备工作

（1）修改applicationContext-service.xml，添加配置

|  |
| --- |
| <context:component-scanbase-package=*"com.pinyougou.page.service.impl"*/> |

（2）pom.xml中引入activeMQ客户端的依赖

|  |
| --- |
| <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-client</artifactId>  <version>5.13.4</version>  </dependency> |

（3）添加spring配置文件applicationContext-jms-consumer.xml

|  |
| --- |
| <!-- 真正可以产生Connection的ConnectionFactory，由对应的 JMS服务厂商提供-->  <beanid=*"targetConnectionFactory"*class=*"org.apache.activemq.ActiveMQConnectionFactory"*>  <propertyname=*"brokerURL"*value=*"tcp://192.168.25.135:61616"*/>  </bean>  <!-- Spring用于管理真正的ConnectionFactory的ConnectionFactory -->  <beanid=*"connectionFactory"*class=*"org.springframework.jms.connection.SingleConnectionFactory"*>  <propertyname=*"targetConnectionFactory"*ref=*"targetConnectionFactory"*/>  </bean>  <!--这个是队列目的地，点对点的 -->  <beanid=*"topicPageDestination"*class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-argvalue=*"pinyougou\_topic\_page"*/>  </bean>  <!-- 消息监听容器 -->  <beanclass=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  <propertyname=*"destination"*ref=*"topicPageDestination"*/>  <propertyname=*"messageListener"*ref=*"pageListener"*/>  </bean> |

### 6.3.3代码编写

创建消息监听类PageListener

|  |
| --- |
| @Component  **publicclass** PageListener **implements** MessageListener {  @Autowired  **private** ItemPageService itemPageService;    @Override  **publicvoid** onMessage(Message message) {  TextMessage textMessage= (TextMessage)message;  **try** {  String text = textMessage.getText();  System.***out***.println("接收到消息："+text);  **boolean**b = itemPageService.genItemHtml(Long.*parseLong*(text));  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

# 7.商品删除-删除商品详细页

## 7.1需求分析

执行商品删除后，同时删除每个服务器上的商品详细页

## 7.2消息生产者（运营商后台）

### 7.2.1配置文件

修改spring-activemq.xml，添加配置

|  |
| --- |
| <!--这个是订阅模式 用于删除页面-->  <bean id=*"topicPageDeleteDestination"*class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-argvalue=*"pinyougou\_topic\_page\_delete"*/>  </bean> |

### 7.2.2代码实现

修改GoodsController.java

|  |
| --- |
| @Autowired  **private** Destination topicPageDeleteDestination;//用于删除静态网页的消息  /\*\*  \* 批量删除  \* **@param** ids  \* **@return**  \*/  @RequestMapping("/delete")  **public** Result delete(**final** Long [] ids){  **try** {  goodsService.delete(ids);  jmsTemplate.send(queueSolrDeleteDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  **return** session.createObjectMessage(ids);  }  });  //删除页面  jmsTemplate.send(topicPageDeleteDestination, **new** MessageCreator() {  @Override  **public** Message createMessage(Session session) **throws** JMSException {  **return** session.createObjectMessage(ids);  }  });  **returnnew** Result(**true**, "删除成功");  } **catch** (Exception e) {  e.printStackTrace();  **returnnew** Result(**false**, "删除失败");  }  } |

## 7.3消息消费者（页面生成服务）

### 7.3.1配置文件

修改pinyougou-page-service的applicationContext-activemq-consumer.xml

|  |
| --- |
| <!--删除页面 -->  <beanid=*"topicPageDeleteDestination"*class=*"org.apache.activemq.command.ActiveMQTopic"*>  <constructor-argvalue=*"pinyougou\_topic\_page\_delete"*/>  </bean>  <!-- 消息监听容器 -->  <beanclass=*"org.springframework.jms.listener.DefaultMessageListenerContainer"*>  <propertyname=*"connectionFactory"*ref=*"connectionFactory"*/>  <propertyname=*"destination"*ref=*"topicPageDeleteDestination"*/>  <propertyname=*"messageListener"*ref=*"pageDeleteListener"*/>  </bean> |

### 7.3.2代码实现

（1）修改pinyougou-page-interface的ItemPageService.java

|  |
| --- |
| /\*\*  \* 删除商品详细页  \* **@param** goodsId  \* **@return**  \*/  **publicboolean**deleteItemHtml(Long[] goodsIds); |

1. 修改pinyougou-page-service的ItemPageServiceImpl.java

|  |
| --- |
| @Override  **publicboolean** deleteItemHtml(Long[] goodsIds) {  **try** {  **for**(Long goodsId:goodsIds){  **new** File(pagedir+goodsId+".html").delete();  }  **returntrue**;  } **catch** (Exception e) {  e.printStackTrace();  **returnfalse**;  }  } |

1. 创建监听类PageDeleteListener

|  |
| --- |
| @Component  **publicclass** PageDeleteListener **implements** MessageListener {  @Autowired  **private** ItemPageService itemPageService;    @Override  **publicvoid** onMessage(Message message) {  ObjectMessage objectMessage= (ObjectMessage)message;  **try** {  Long[] goodsIds = (Long[]) objectMessage.getObject();  System.***out***.println("ItemDeleteListener监听接收到消息..."+goodsIds);  **boolean**b = itemPageService.deleteItemHtml(goodsIds);  System.***out***.println("网页删除结果："+b);  } **catch** (JMSException e) {  e.printStackTrace();  }  }  } |